Excursions into Algorithms
Algorithms undoubtedly play a central role in modern **industry** and **life**.
However, algorithm is not a temporary fad. It has a **LONG HISTORY**.

*Persian mathematician, astronomer, and geographer*

**al-Khwarizmi, Latinized as *Algorithmi*. (780-850)**

Presented the first systematic solution of linear and quadratic equations using geometric justifications.
However, algorithm is not a temporary fad. It has a LONG HISTORY.

Clay tablets from the Old Babylonian period (1830–1531 BC)

Topics including fractions, algebra, quadratic and cubic equations and the Pythagorean theorem.
However, algorithm is not a temporary fad. It has a LONG HISTORY.

Alan Turing (1912 – 1954)

The father of Computer Science

Presented techniques (Bombe) for speeding the breaking of German ciphers (Enigma). Turing machine, Turing Award.
However, algorithm is not a temporary fad. It has a LONG HISTORY.

GREAT LEAPS

All the major search engines try to return to the user the most relevant results from the most trusted sources.

Google PageRank
Big Data, Deep Learning, Algorithms,...

However, algorithm is not a temporary fad. It has a **LONG HISTORY**.

We still are searching for efficient/optimal algorithms for fundamental/important problems.

We also let Machine Learning and AI solve notoriously difficult problems.
ARTIFICIAL INTELLIGENCE
Mimic the cognitive functions of humans

MACHINE LEARNING
Neural Network
a series of algorithms modeled after the human brain.

DEEP LEARNING
Multiple layers of NN
Exposure to large quantities of items in order to be trained
Algorithm is

- unambiguous specification of how to solve a class of problems.
- procedure or rule for solving a problem, based on conducting a sequence of specified actions.
- a computer program is an algorithm.

Success Stories

- Link Analysis
  - Rank/WebSearch
- Graph Traversal
  - DFS/BFS/MST
- Sorting
  - Bubble/Merge/Quick
- Binary Searching
  - logarithmic time
- Hashing
  - constant time
- Data Compression
  - Huffman encoding
Good Algorithms

Efficiency
Correctness

Good algorithms deliver the correct answer in the most efficient manner.

How many basic operations does it execute in terms of the input size?

assuming a basic operation (+, −, *, assign, ...) takes $O(1)$ time,

- **Bubble sort** takes $O(n^2)$ time vs. **Merge sort** takes $O(n \log n)$ time.
- **Linear search** takes $O(n)$ time vs. **Binary search** takes $O(\log n)$ time.

* Asymptotic complexities: $O(\cdot), \Omega(\cdot), \Theta(\cdot)$ on functions of input size $n$.

*asymptotic upper (O), lower bounds (Omega); worst case complexity (theta)
\[ f(n) = 2^n \]
\[ f(n) = n^2 \]
\[ f(n) = n \log_2 n \]
\[ f(n) = n \]
\[ f(n) = \log_2 n \]
Good Algorithms

Efficiency
Correctness

Good algorithms deliver the correct answer in the most efficient manner.

How many basic operations does it execute in terms of the input size?

assuming a basic operation (+, −, *, assign, ... ) takes $O(1)$ time,

Small data - Do not make much difference.
Big data/number - Substantial/significant difference in running times.

For 1,000,000 sorted data stored in an array,

**Linear search** performs almost 1,000,000 comparisons.

**Binary search** performs at most $\log 1,000,000 \leq 20$ comparisons.
BIG & Highly complex DATA

Video/Images of high frequency/resolution from Scanners, LiDAR sensors, Big graphs (Web pages, Facebook, Twitter,...)

2.2 million 3D points per second
All the major search engines try to return to the user the most **relevant results** from the most **trusted sources**.
Google Search wanted to measure relative importance of websites (link popularity).

- A hyperlink to a page counts as a vote of support.
- PageRank of a page is defined recursively on the number and PageRank metric of all pages that link to it ("incoming links").
- A page linked to by many pages with high PageRank receives a high rank.
We are searching for a solution (path, tree, matching, etc.) among an exponential population of possibilities:

- $n$ boys and $n$ girls can be matching in $n!$ different ways.
- A graph with $n$ nodes has $n^{n-2}$ spanning trees.
- A typical undirected graph has exponential number of paths from $s$ to $t$.

All these problems can be solved in exponential time by checking three all candidates one by one, but this takes too much time to be useful in practice.

So far we have seen success stories, algorithmic techniques that defeat the specter of exponentiality.

- Greedy algorithms - $O(|E| \log |V|)$ time for minimum spanning trees.
- Dijkstra’s algorithm - $O((|V| + |E|) \log |V|)$ time for $s-t$ shortest path.
- Dynamic programming - $O(n^3)$ time for chain matrix multiplication.
- Linear programming - $O(n^3)$ time for bipartite matching.

But there still are search problems for which the fastest algorithms we know are all exponential - not substantially better than an exhaustive search.
P and NP

P stands for “polynomial” while NP stands for “nondeterministic polynomial time”.

A solution to any search problem can be found and verified in polynomial time by a nondeterministic algorithm.

(A nondeterministic algorithm may exhibit different behaviors on different runs even for the same input. Nondeterministic poly-time means an algorithm may run in poly-time or exponential time depending on the choices it makes during execution.)

Complexity classes include

- **NP-complete**: NP but $O(n^k)$-time algorithm unlikely.
- **Undecidability**: No algorithm that always gives the correct answer.
  
  Ex. The halting problem (determining whether a Turing machine halts for an arbitrary program and a finite input).

*NP-hard & NP

- A problem H is NP-hard when every problem L in NP can be reduced in polynomial time to H; finding a polynomial time algorithm to solve any NP-hard problem would give polynomial time algorithms for all the problems in NP, so it is at least as hard as the hardest problems in NP.

- under the assumption that $P \neq NP$
Satisfiability (SAT)

An instance of SAT is a Boolean formula in CNF (conjunctive normal form):

\[(x \lor y \lor z)(x \lor \bar{y})(y \lor \bar{z})(z \lor \bar{x})(\bar{x} \lor \bar{y} \lor \bar{z})\]

A collection of clauses (the parentheses), each consisting of the disjunction of several literals (a Boolean variable or the negation of one).

A satisfying truth assignment is an assignment of true or false to each variable so that every clause contains a literal whose value is true.

SAT problem. Given a Boolean formula in CNF, either find a satisfying true assignment or report “none exists”.

An exhaustive search on \(n\) variables - \(2^n\) possible assignments!

SAT is a typical search problem. We are given an instance \(I\), and we are asked to find a solution \(S\).

Property of a search problem. Any proposed solution \(S\) to an instance \(I\) can be quickly checked for correctness. That is, there is a polynomial-time algorithm that takes as input \(I\) and \(S\), and decides whether or not \(S\) is a solution for \(I\).
Satisfiability (SAT)

Researchers over the past 50 years have tried to find efficient ways to solve it, but without success. The fastest algorithms we have are still exponential on their worst-case inputs.

“\[\text{I can’t find an efficient algorithm, but neither can all these famous people.}\]

But, interestingly, there are two natural variants of SAT for which we have good algorithms.

- **Horn formula.** All clauses contain at most one positive literal. A greedy algorithm can find a solution in linear time.
- **2SAT.** All clauses have only two literals. Can be solved in linear time by finding the strongly connected components.
Independent Set problem. Given a graph and a goal $g$, find $g$ vertices that are independent, that is, no two of which have an edge between them.

Vertex Cover problem. Given a graph and a budget $b$, find $b$ vertices that cover (touch) every edge. Vertex Cover is a special case of Set Cover, in which we are given a set $E$ and several subsets of it, $S_1, \ldots, S_m$, along with a budget $b$, and we aim to select $b$ of these subsets so that their union is $E$. Let $E$ be the set of the edges of a graph, and $S_i$ be the set of the edges adjacent to vertex $v_i$ for each vertex $v_i$. 3D Matching is also a special case of Set Cover.

Clique problem. Given a graph with $n$ vertices and a budget $b$, find a set of $b$ vertices such that all possible edges between them are present.
Independent Set, Vertex Cover, Clique

Claim. Vertex-Cover $\equiv_P$ Independent-Set.

*polynomial time reduction

Proof. We show $S$ is an independent set iff $V \setminus S$ is a vertex cover.

Let $S$ be any independent set. Consider an arbitrary edge $(u, v)$. Since $S$ is an independent set, we have $u \notin S$ or $v \notin S$. This implies that $u \in V \setminus S$ or $v \in V \setminus S$. Thus, $V \setminus S$ covers $(u, v)$.

Let $V \setminus S$ be any vertex cover. Consider two nodes $u \in S$ and $v \in S$. Since $V \setminus S$ is a vertex cover, the graph does not have edge $(u, v)$. Thus, no two nodes in $S$ are joined by an edge, which implies that $S$ is an independent set.
**NP-Complete Problems**

The world is full of search problems, some of which can be solved efficiently, while others seem to be very difficult, as depicted in the following table.

<table>
<thead>
<tr>
<th>Hard problems (NP-complete)</th>
<th>Easy problems (in P)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3SAT</td>
<td>2SAT, HORN SAT</td>
</tr>
<tr>
<td>TSP</td>
<td>MST</td>
</tr>
<tr>
<td>TSP</td>
<td>MST</td>
</tr>
<tr>
<td>Longest Path</td>
<td>Shortest Path</td>
</tr>
<tr>
<td>3D Matching</td>
<td>Bipartite Matching</td>
</tr>
<tr>
<td>Independent Set, Vertex Cover, Clique</td>
<td>Independent Set on trees</td>
</tr>
<tr>
<td>Rudrata (Hamiltonian) Path</td>
<td>Euler Path</td>
</tr>
<tr>
<td>Balanced Cut</td>
<td>Minimum Cut</td>
</tr>
</tbody>
</table>

The problems on the left are all difficult for the same reason!
**P and NP**

**Property of a search problem.** Any proposed solution $S$ to an instance $I$ can be **quickly checked** for correctness. There is an efficient checking algorithm $C(I, S)$ that takes time polynomial in $|I|$.

$\text{NP : the class of all search problems.}$

$\text{P : the class of all search problems that can be solved in polynomial time. Most of search problems we have considered belong to P.}$

$\text{P = NP ?}$

Most researchers believe $P \neq NP$. But no proof. Then why they believe so? There are hard problems, famously unsolved for decades and centuries. But on what evidence do we believe there’s no efficient algorithms for them?

**Reduction provides some evidence.**

The hard problems in the left side of the table are the hardest search problems in NP, and all are the same problem (by reduction).